

Landscape Art Adventures

```
(ns opencv3.cartoon3
  (:require
   [opencv3.core :refer :all]
   [opencv3.utils :as u]))
```

```
nil
```

```
(def img
  (-> "https://cdn.theculturetrip.com/wp-content/uploads/2016/01/canals2.jpg"
    u/mat-from-url
    (u/resize-by 0.2)))
(u/mat-view img)
```

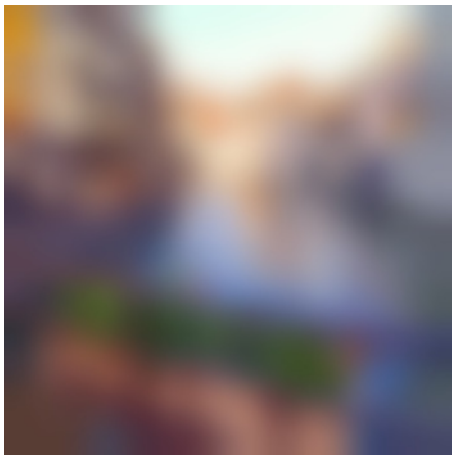


smoothing the picture using a bilateral filter

```
(def factor 4)
(def output (new-mat))
(def work (clone img))

(dotimes [_ factor] (pyr-down! work))
(bilateral-filter work output 9 9 7)
(dotimes [_ factor] (pyr-up! output))

(u/mat-view output)
```



detect and enhance edges

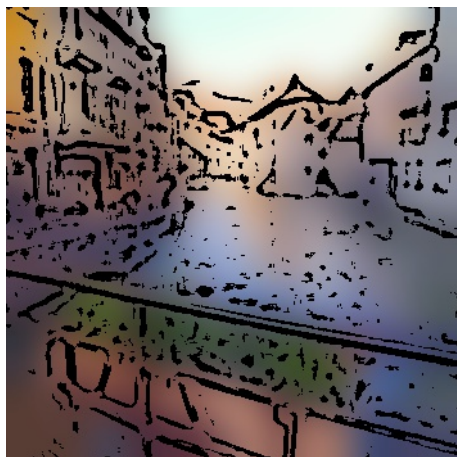
- Reduce noise using a median filter
- Create an edge mask using adaptive thresholding

```
(def edge
  (-> img
    clone
    (resize! (new-size (.cols output) (.rows output)))
    (cvt-color! COLOR_RGB2GRAY)
    (median-blur! 7)
    (adaptive-threshold! 255 ADAPTIVE_THRESH_MEAN_C THRESH_BINARY 9 7)
    (cvt-color! COLOR_GRAY2RGB)))
(u/mat-view edge)
```



Combine color image with edge mask

```
(let [result (new-mat) ]  
  (bitwise-and output edge result)  
  (u/mat-view result))
```

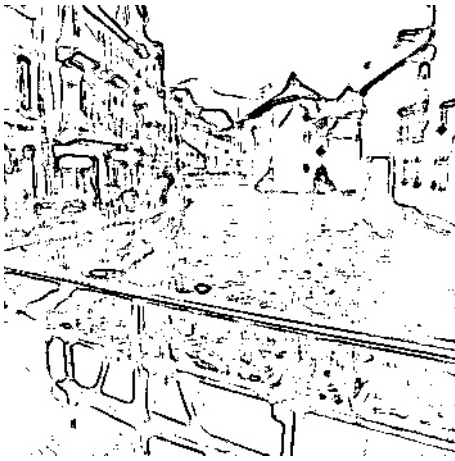


Playing with contours

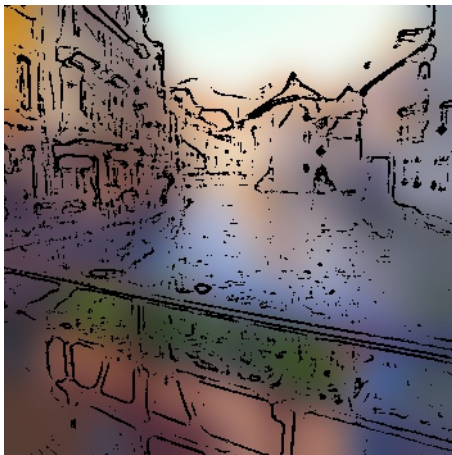
Of course you can play with the contours size and numbers. The `adaptive-threshold!` does that with its last two parameters.

The **edges-thickness** parameter controls how thick the contours will be, and the **edges-number** controls how many of them should be drawn.

```
(def edges-thickness 5)  
(def edges-number 5)  
(def edge  
  (-> img  
    clone  
    (resize! (new-size (.cols output) (.rows output)))  
    (cvt-color! COLOR_RGB2GRAY)  
    (median-blur! 7)  
    (adaptive-threshold! 255 ADAPTIVE_THRESH_MEAN_C THRESH_BINARY edges-thickness edges-number)  
    (cvt-color! COLOR_GRAY2RGB)))  
(u/mat-view edge)
```



```
(let [result (new-mat) ]
  (bitwise-and output edge result)
  (u/mat-view result))
```



turning all this in functions

```
(defn smoothing!
  [img factor filter-size filter-value]
  (let [ work (clone img) output (new-mat)]
    (dotimes [_ factor] (pyr-down! work))
    (bilateral-filter work output filter-size filter-value)
    (dotimes [_ factor] (pyr-up! output))
    (resize! output (new-size (.cols img) (.rows img)))))

(defn edges!
  [img e1 e2 e3]
  (-> img
    clone
    (cvt-color! COLOR_RGB2GRAY)
    (median-blur! e1)
    (adaptive-threshold! 255 ADAPTIVE_THRESH_MEAN_C THRESH_BINARY e2 e3)
    (cvt-color! COLOR_GRAY2RGB)))

(defn cartoonize!
  [ img s1 s2 s3 e1 e2 e3]
  (let [ output (smoothing! img s1 s2 s3) edge (edges! img e1 e2 e3)]
    (bitwise-and output edge output)
    output))
```

```
#'opencv3.cartoon3/cartoonize!
```

```
(->
  "resources/landscape/landscape-nature-sky-blue.jpg"
  imread
  (u/resize-by 0.2)
  (cartoonize! 6 9 7 7 9 11)
  (u/resize-by 0.5)
  (u/mat-view ))
```



```
(->  
  "resources/landscape/amazing-beautiful-beauty-blue.jpg"  
  imread  
  (cartoonize! 5 9 7 7 7 5)  
  (u/resize-by 0.25)  
  (u/mat-view))
```



```
(->  
  "resources/landscape/amazing-beautiful-beauty-blue.jpg"  
  imread  
  (u/resize-by 0.25)  
  (cartoonize! 5 9 7 7 7 5)  
  (u/mat-view))
```

